AN IMPLEMENTATION OF THE FORTH LANGUAGE ON HEATH H-89 MICROCOMPUTER

A THESIS

SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

THE DEGREE OF MASTER OF SCIENCE

BY

TZU-LIN L. HUANG

DEPARTMENT OF MATHEMATICAL SCIENCE

ATLANTA UNIVERSITY

ATLANTA, GEORGIA

DECEMBER, 1981

ABSTRACT

COMPUTER SCIENCE

HUANG, TZU-LIN L. B.L. NATIONAL TAIWAN UNIVERSITY, 1975

AN IMPLEMENTATION OF THE FORTH LANGUAGE ON HEATH H-89 MICROCOMPUTER

Advisor: Dr. Benjamin J. Martin

Thesis dated December 1981

The primary intent of this thesis is to discuss how to use the Forth language on the Heath H-89 microcomputer. Forth is a programming language with a small, but fast-growing, enthusiastic user community. Forth was invented by Mr. Charles H. Moore in 1973.

Forth is a very compact language which only needs a small memory storage. It is also a threaded language which starts with a few fundamental subroutines written in machine language.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I wish to express my gratitude to those who have helped, assisted, and encouraged me in all my studies and researches while I was in attendance at Atlanta University. My deepest appreciation goes to

Dr. Benjamin J. Martin

Dr. Johnny L. Houston

Dr. Nazir A Warsi

My Parents, Brothers and friends

All the classmates in the Department of Mathematical Sciences

And especially to God, who is faithful and who always brings me through.

List Of Figures

CHAPTER I

INTRODUCTION

The Forth Development History

Forth is a programming language which was invented by Mr. C. H. Moore, who along with E. R. Rather constructed the modern Forth system at the National Radio Astronomy Observatory. When they invented this language, their intention was simply to make themselves more productive programmers. They figured that in 40 years a very good programmer could write forty programs. They wanted to write more programs than that and needed a tool to help them. When they worked on programs that ranged from satellite orbits to chromatography to business systems, they developed Forth in line with this goal. Since that time they have been able to work at ten times their original rate.

The first Forth version was designed for the IBM 1130. During that time Mr. Moore worked directly on an IBM 1130 system, interfacing with an IBM 2250 graphics display. The IBM 1130 let the programer, for the first time, totally control the computer interactively. At that time, Mr. Moore found out that he used the IBM 1130 in FORTRAN in 32k bytes could draw pictures on the IBM 2250 graphics disply fairly slowly; and Forth, in 8k bytes,

could draw three-dimentional moving pictures on the IBM 2250 graphical display very fast.

In the first version of Forth, the high-level or colon definitions were not yet compiled. The text interpreter reinterpreted the text as the Basic language acts today. Later, Mr. Moore implemented Forth on a Univac 1180, interfacing it with their COBOL compiler.

The mordern Forth was coded in FORTRAN. Shortly thereafter it was recoded in Forth. The additions were the return stack and the dictionary. The dictionary was in the form of linked list.

The use of modern Forth occurred when it was written for a Honeywell H-316 at the NRAO (National Radio Astronomy Observatory). In 1971, Mr. Moore wrote a radio-telescope data-acquisition program that led him to put the compiler on the Forth system. An average of five characters per word could be replaced by twp bytes per word.

Since then, as Forth's usage increased in astronomical reserach, the user community recognized the need for a common standardization. In 1977, an ad hoc team within the International Astronomical Union hosted by Peak National Observatory, tried to standardize the Forth Language. This group produced the document AST.1, a

glossary of definitions for the minicomputer versions.

In 1979, the Forth Interest Group published an implementation model of Forth called Fig-Forth. Applications were transported among computer using the Intel 8080, Digital Equipment LSI-11, Motorola 6800, National Semiconductor PACE, and the MOS Technology 6502.

A Forth Standards Teams met again at Catalina Island in October, 1979. The general goal was to increase the portability of Forth programs by language specifications in addition to the current glossary definition. During this meeting, they created the new Forth version called Forth-79. Forth-79 offers a uniform vehicle for interchange of Forth language. The reward of standards work is: less cost for creating and maintaining specific application programs, less cost for personnel training, and less cost for the host operating system itself.

Forth has been implemented on the Burroughs 5500; the IBM 1130; the Univac 1108; the Honeywell 316; the IBM 360; the Data General Nova; the PDP-11; the Interdata; the CDC-6400; the Computer Automation LSI 4; and the RCA 1802. Forth has been programmed in Fortran, Algol, PL/1, COBOL, Assembler, and Forth.

The Forth Usage In The Real World

Forth is being used in the real world in many areas. Elicon Inc. of Bred, California is using Forth software to drive the computer-controlled cameras. Magicam Inc. is in the process of converting control of its master-slave camera pair from an analog computer to a digital computer running Forth software. Allen Test Products of Kalamazoo, Michigan has developed an ignition analyzer to use in service stations and automobile repair shops that analyze the behavior of automobile ignition systems and displays both diagnostic and corrective information. Atari Inc. is using Forth in the Coin-Operated Division, which develops and markets the stand-alone games found in pinball arcades, and in restaurants. A 6502 based development system employs Forth software to debug and test arcade circuit boards.

Forth is used in several aerospace applications. A Forth-like language, IPS, is uses in the orbiting Earth in an amateur radio satellite called the OSCAR Phase III.

The Forth Overview

Forth is characterized by five major elements: dictionary, stack, interpreters, assembler, and virtual memory. The dictionary is a threaded list of variables, each of which defines a word in the vocabulary. The dictionary is extensible, and redefinable growing toward the memory.

Forth has two push-down stacks. These provide the primary communication between routines as well as an efficient mechanism for control logical flow. A stack normally contains items one computer word in length which may be addresses, numbers, or other objects.

Forth has two levels of interpretation. The first level is a text interpreter. It works in a conventional manner, parsing text strings that come from terminals and looking up each word in the dictionary. The second level of interpretation is an address interpreter. Most dictionary definitions contain addresses of previously defined words that are to be executed.

Forth includes a resident assembler, which allows the programmer to define words that will cause the specified mechine instructions to be executed. Forth uses the virtual memory concept to store the program or data. The memory is devided into segments called blocks or screens.

Each block or screen has the number to identify itself. There is no file name for each Forth program. The Forth system uses the screen number as identification.

Most Forth systems consists of th editor, source listing, disk copy, I/O control, operating system, and compiler.

CHAPTER II

THE STURCTURE OF FORTH

The Dictionary And User-Defined Words

The central element of the Forth system is the "word". A Forth word is similar to a subroutine or procedure in other languages. Executing, or calling, a word causes a definite sequence of actions to be performed. The reason for calling a Forth routine a "word" is that it nearly always has a name that is known to the keyboard interpreter: it can be executed simply by typing its name. Words are defined in the dictionary which , like English dictionaries, is a table of word-names and their definitions. Two types of definitions occur in the Forth dictionary. Words may be defined in terms of other words that are defined earlier, or words may be defined by a sequence of machine language instructions.

All programs written in Forth appear as words or collections of words in the dictionary. Forth dictionaries are organized as threaded lists, each of those elements is the definition of a word. The simplest list structure has a single linear thread connecting the Forth words in the sequence of their definition. The

dictionary is split into 16 threads or branches. The branch in which a word appears is a function of its name.

The head of the list is defined by a 16-element vector of pointers. These pointers aim at the most recently defined word in each branch. A field in which each word definition in turn points to the previous word appears is under control of the user. The words VOCABULARY and DEFINITIONS control the branching.

The defining word compiles the words that follow it in a definition, which is then added to the dictionary. Each Forth dictionary definition consists of two parts : a head and a body. The head contains system-internal information including a name field and a code field. The name field contains the name of the word. The code field contains a pointer to the instructions that will be executed when the word is executed.

In Forth, a word is any string of characters bounded by spaces. The characters that represent arithmetic operators or punctuation marks can be words if they are bounded by spaces. For example, each of the following strings is a word in the Forth system:

EMIT BEGIN UNTIL FLUSH ! * OVER

For definitions compiled by : (colon"), the code

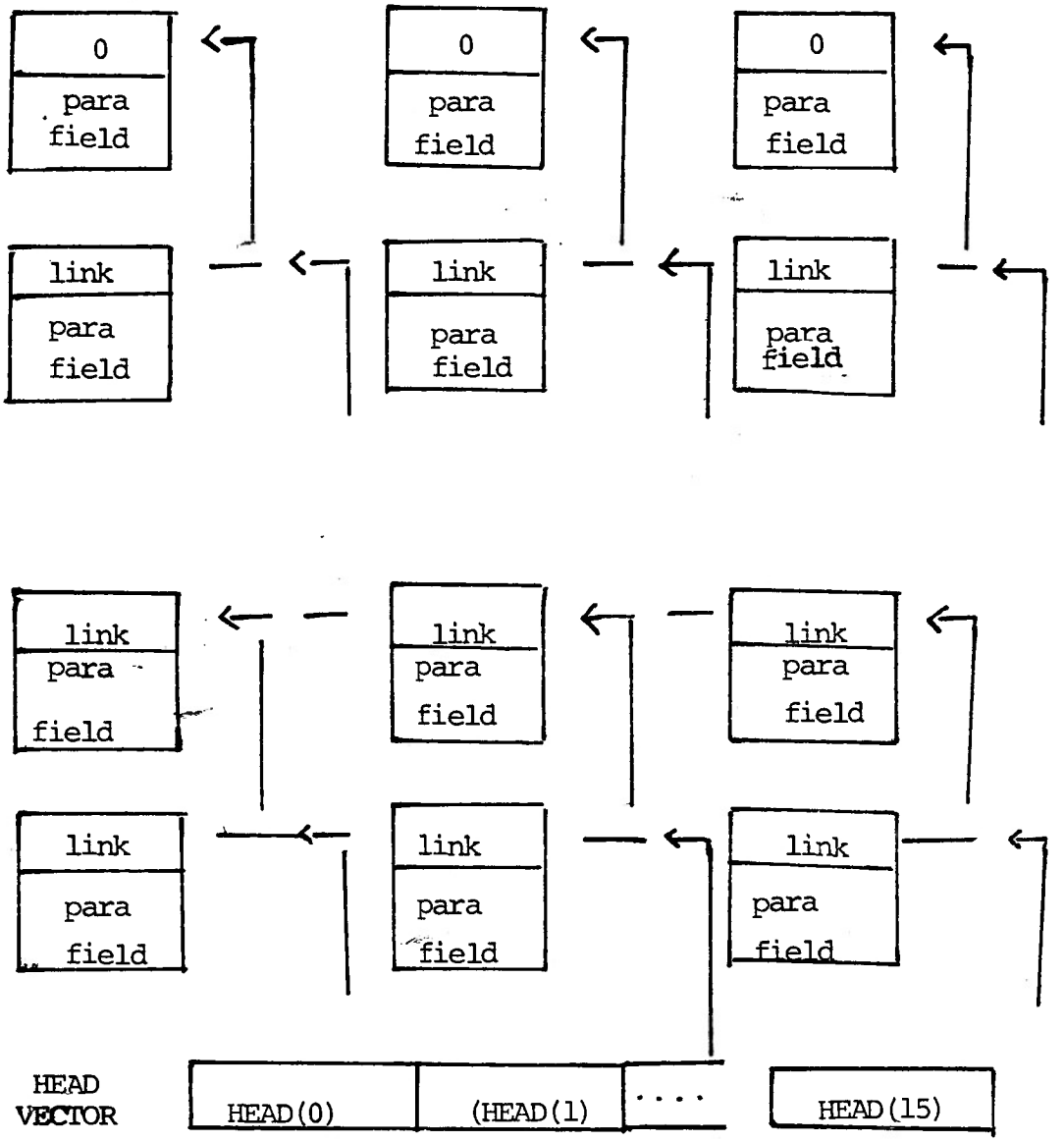


Fig. 1. Dictionary Organization

field points to a procedure that begins the execution of the words referenced in the definition. The body of definition, called parameter field, is a series of addresses that point in order to each Forth word in the definition.

Words are added to the dictionary by "defining a word" of which the most common is ":" (colon). The execution of ":" causes a dictionary entry to be constructed for the following word. The definition of this new word, in the form of addresses of previously defined words, will also be placed in the dictionary. The definition is terminated by ";" (semicolon). For example:

```
: CASE FLUSH SWAP OPEN ROT EMIT @ ;
```

could be a new word CASE. Previously defined words such as "FLUSH", "SWAP", "OPEN", "ROT", "EMIT", "@" perform the various component operations. CASE becomes the new word in the dictionary. This word is terminated by the symbol ";" (semicolon). After we define this word CASE, the system will execute and compile. In the execution state, the text interpreter operates executing words as they are found in the input text. The interpreter will put the address of FLUSH, address of SWAP, address of OPEN, address of ROT, address of EMIT, and address of "@" on the

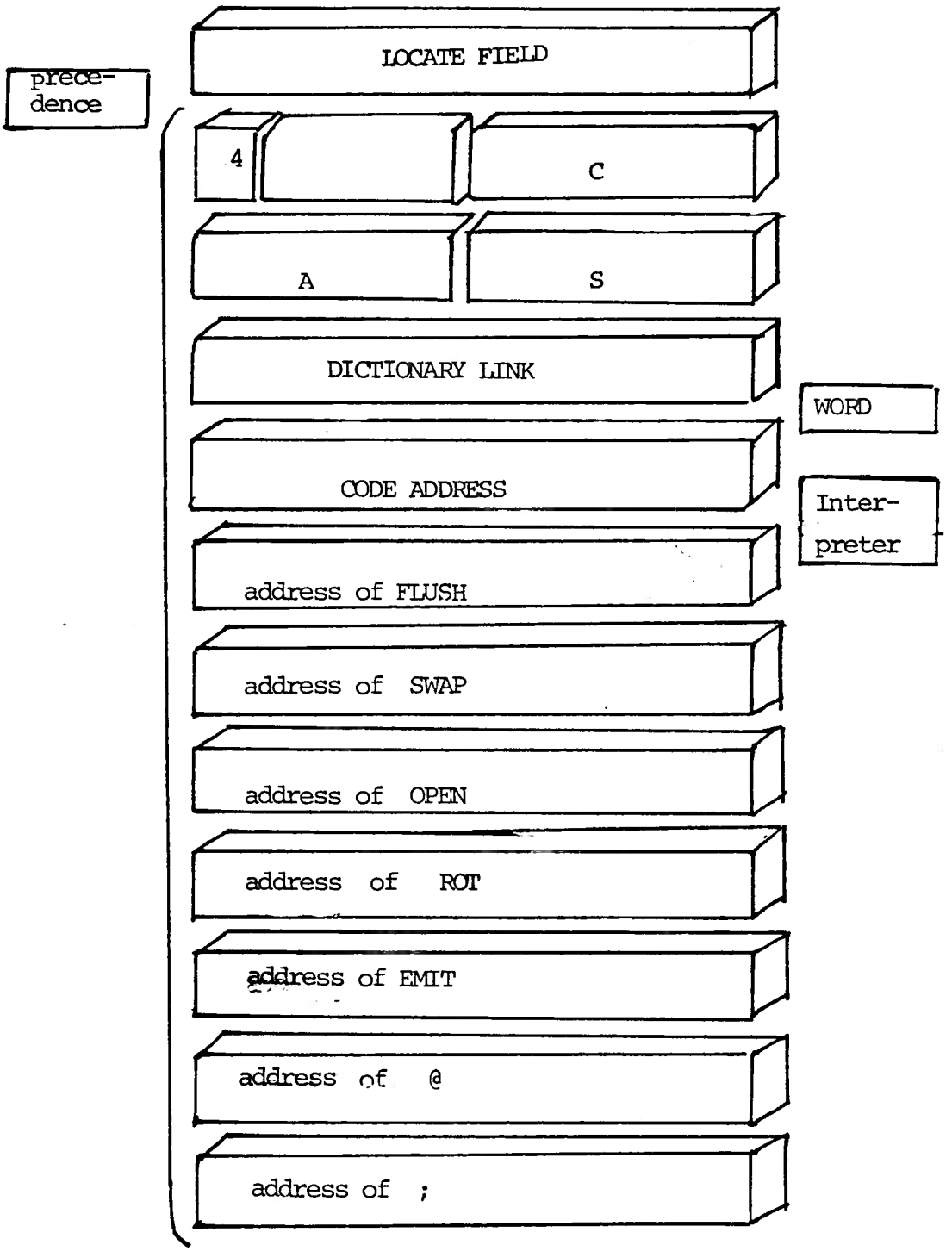


Fig. 3. Dictionary Entry for the word CASE

dictionary entry, then the word ":" (colon) changes the state to compilation; the word name CASE is placed in the next available dictionary locations in the correct dictionary format. The link field is set to point to the last-defined word in the same dictionary branch, and the head pointer is set to point to the new entry. After defining the new word CASE, the user can save this word in the memory and CASE becomes part of the dictionary. He can use the word CASE in any program or statement as the word is defined by the system. When the user wants to invoke and execute the word CASE, he types the word CASE and the system will interpret and automatically execute it.

In Forth, there are several linked chains. We call these chains Vocabularies.

1. FORTH Contains all the basic definitions.
2. ASSEMBLER Contains the assembler mnemonics.
3. EDITOR Contains the commands for entering source text.

Invoking a vocabulary name specifies which linked list will be searched and in what order. For example, the word I in the Forth vocabulary supplies a loop index. If you invoke the Editor vocabulary, the word I inserts text.

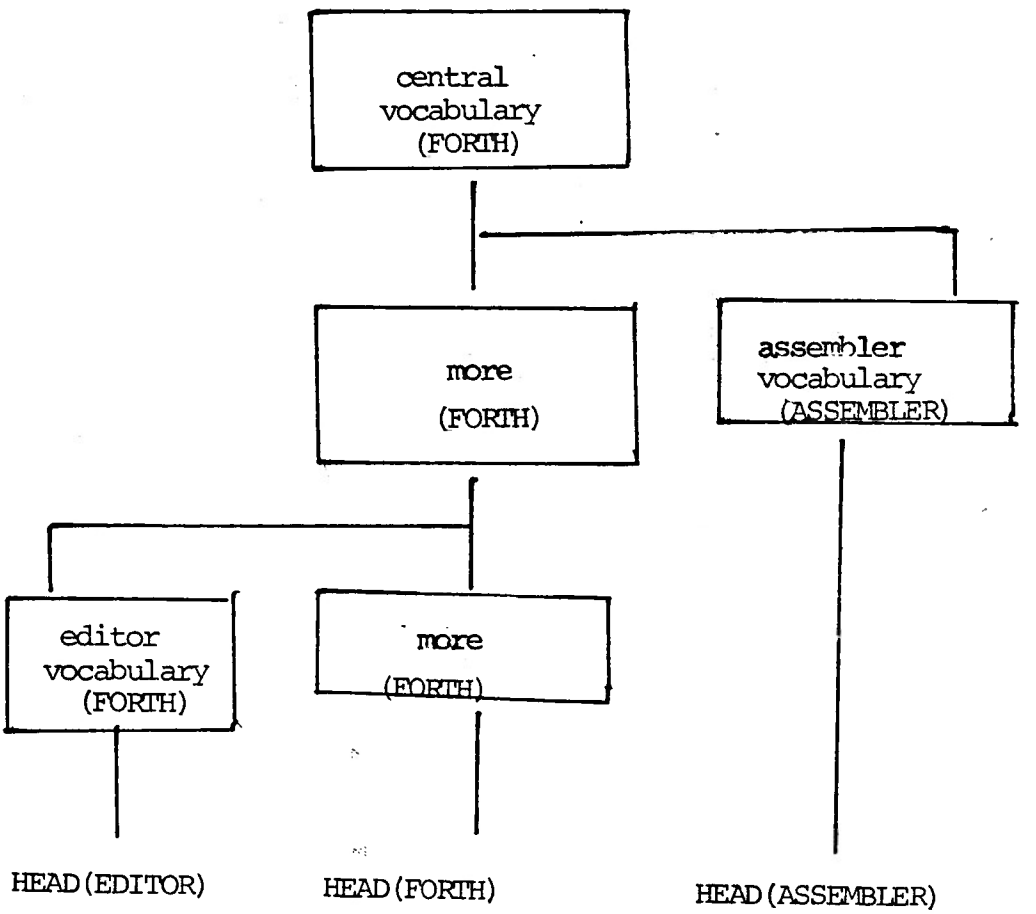


Fig. 2. Forth Vocabulary Structure

Which definition will be executed depends on which vocabulary is searched first. After invoking Editor, the order is Editor, then Forth. When using the Forth vocabulary, Forth will be searched and then, on most system the Editor.

The concept of separate vocabularies makes it possible for different words with the same name to co-reside in memory. CONTEXT is the name of the user variable that contains the index of the vocabulary that will be searched first to find a word.

The Stack

Forth uses the stack to input, output, and loop controls. The stack is "push-down", also called Last-In First-Out (LIFO) list. This stack is a way of storing data such that the most recently stored items are immediately accessible. Forth calculates and returns numbers similar to the Hewlett-Packard Calculators. To see how a LIFO stack works, type the following at the terminal.

```
25 36 17 9 CR ok
```

The system's response, ok, indicates that all commands up to the carriage return have been executed successfully.

The four values have been pushed onto the stack. The first value entered (25) is on the bottom, the last (9) is on top.

Now type

The computer will respond:

```
9 17 36 25
```

The "." (dot) is a Forth word which removes the top value from the stack and prints it at the terminal. The first "." prints the 25 and leaves the 36 on the top of the stack, etc. The last value entered is the first one removed.

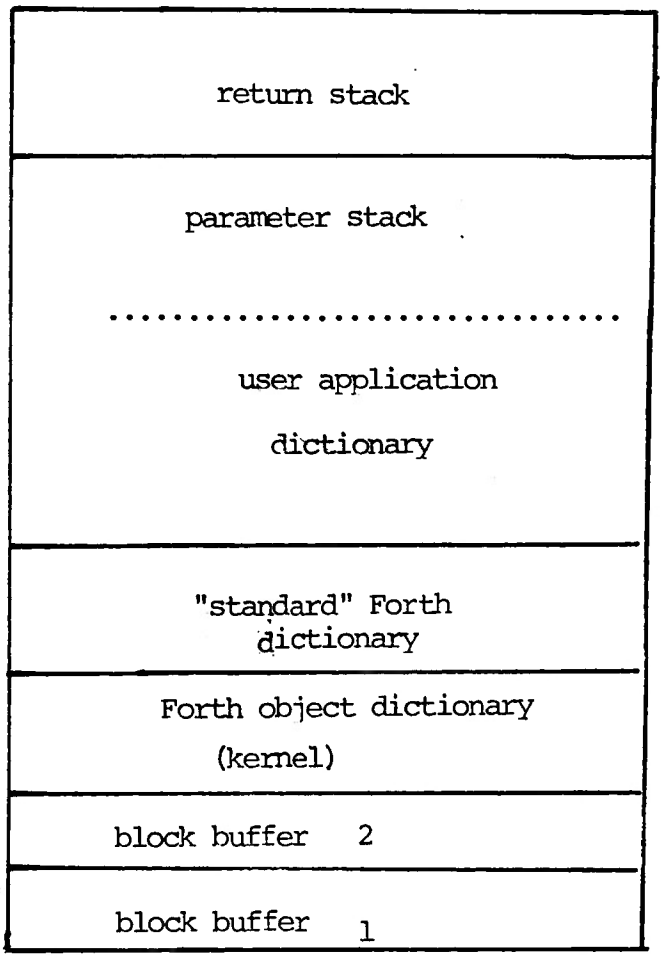
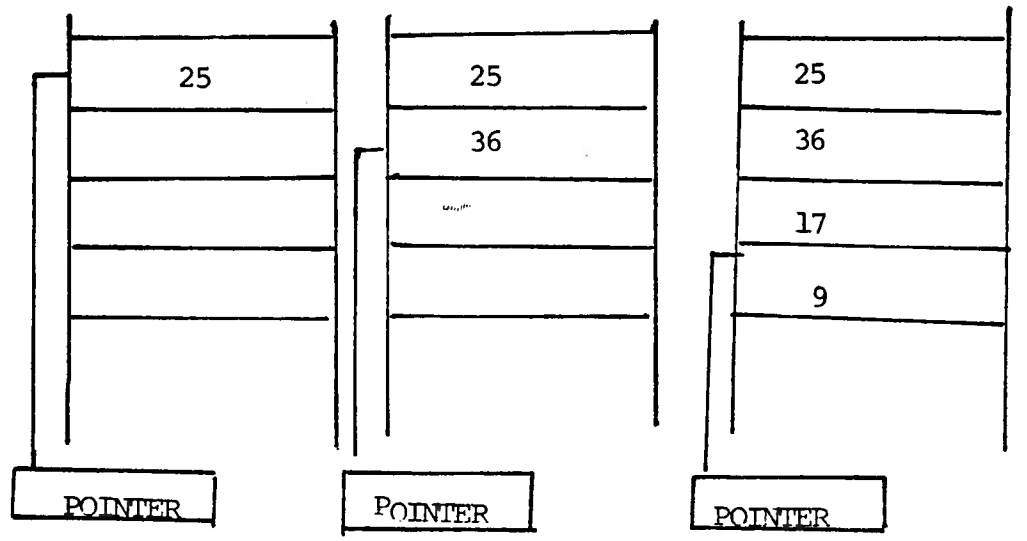


Fig. 4. Memory Layout of a Forth System



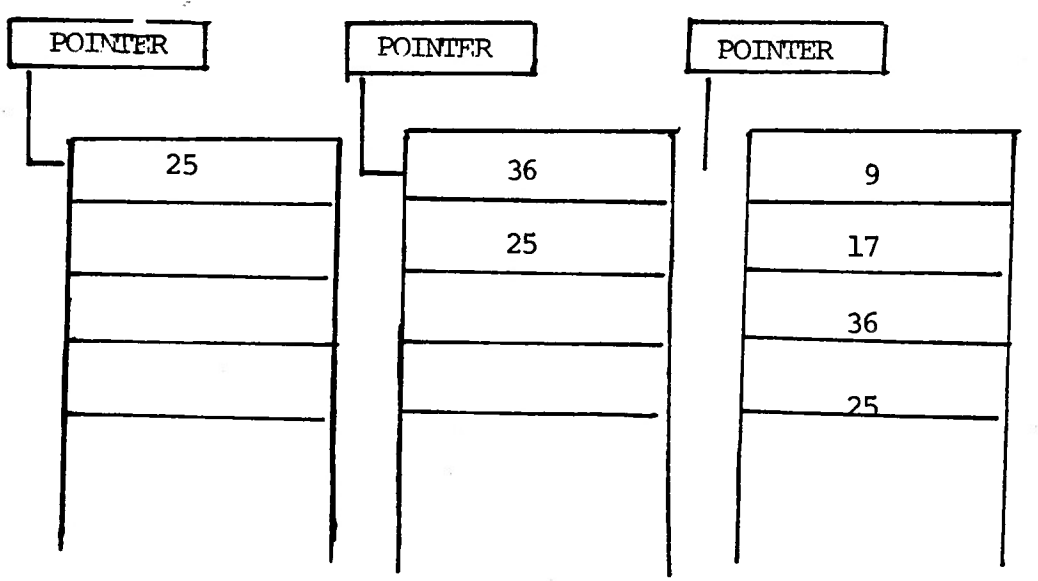


Fig. 5. Stack Pointer Viewpoint

Forth's extreme modularity is faciliated by its use of the stack to pass data from word to word. To accomodate this feature, Forth uses postfix notation rather than the more familiar algebraic or infix notation. In Forth the arithmatical operators always work on the value present on the stack. For example, the phrase

99 34 - will push the value 99 onto the stack, then the value 34, leaving the 99 below it. The "-" (minus) will subtract the top value from the second value and leave only the result (65).

Postfix (Forth)	Infix
4 3 -	4 - 3
25 3 9 */	(25 * 3) / 9
25 3 9 * /	25 / (3 * 9)
3 9 /	3 / 9
42 5 *	42 * 5

Numbers And Arithmetic

In Forth, arithmetic is performed on integers and fixed-pointed fractions rather than on floating-point numbers. The use of integers takes advantage of the speed of integer arithmetic operations in most minicomputers and microprocessors.

Forth recognizes either signed or unsigned integer values. Negative values are stored as two's complement numbers. Single-precision (16-bit number representation) tolerates a range of 0 to 65535 for unsigned values, and -32768 to 32767 for signed values.

The Forth word "." (dot) prints the signed integer at the terminal. The word "U." (U-dot) prints the value as an unsigned integer.

The following examples might show some of the differences by using the different number representation.

The User Enters	The Computer Responds:
32 .	32 ok
96 .	96 ok
32767 .	32767 ok
32768 .	-32768 ok
32768 U.	32768 ok

The last two examples above show what happens when the number exceeds 32767 as a signed sixteen-bit integer: the sign bit is set to one, and "." interprets the value as a negative integer. Printing the value as an unsigned integer by using the word "U." produces the expected result.

In Forth any value that is unpunctuated will be converted as single-precision, otherwise as double-precision (32-bit number representation). a double-precision number is converted as two consecutive cells on the stack, the high-order cell on top. The following examples show the differences of the number representation.

The User Enters:	The Computer Responds:
65,536 . .	1 0 ok
76.345 . .	1 10809 ok
32.110 . .	0 32110 ok
31.001 . .	0 31001 ok

The Forth word "D." outputs a 32-bit value.

The User Enters:	The Computer Responds:
65,536 D.	65536 ok
76.345 D.	76345 ok

```
32.110 D.                    32110 ok
31.001 D.                    31001 ok
```

In Forth, all number conversions will use base ten (decimal) for both input and output. We can easily change the base by using one of the following commands:

```
OCTAL(set the base to 8)
HEX  (set the base to 16)
DECIMAL (set the base to 10)
```

The user can use any number base by using the phrase:

```
n BASE !
```

For example:

```
: BINARY 2 BASE ! ;
```

The above phrase will define the word BINARY.

The User Enters:	The Computer Responds:
DECIMAL 24567 BINARY .	101111111110111 OK

Constants, Variables and Arrays

In Forth, just like other languages, we can assign a name to a constant. The user can only name an integer. The user can not name a real number because there is no float point operation in Forth. For example, if we want to assign the value 60 to the name AGE just type

```
60 CONSTANT AGE
```

It will create the new word AGE and assigning it the value 60. The phrase "90 AGE *" will have the result of 60 * 90, that is 5400.

The value which may change frequently is called a variable. Every variable in Forth must be declared by the programmer first, then he can use it later. The user can use any characters to name the variable. For example:

```
VARIABLE POSITIVE
```

The user creats a variable name POSITIVE. The computer will assign a address to the variable POSITIVE. Whenever invoking a variable, it will place the address on the stack. If the user assigns the content of the address of POSITIVE, he can type the phrase:

```
987 POSITIVE !
```

The word "!" (store) is used to store a value a location. Currently the contents of the address named

POSITIVE has the value 987. If he wants to add a number to the variable POSITIVE, just types

56 POSITIVE +! will add 56 to the content of address POSITIVE and store back to the POSITIVE. Now the contents of the address is 987 + 56 which is 1043.

In Pascal and Fortran, the linear and higher-dimentional arrays are used. In Forth, there is no array structure in the original dictionary. The user can use the word ALLOT to assign the extra space in memory. For example: 1 VARIABLE AGE 98 ALLOT
will create an extra 99 spaces in memory.

Virtual Memory

Forth implements virtual memory by considering disks to be organized in numbered blocks, from zero to the capacity of the disk controller. These blocks are read into buffers, each of which provides 1024 bytes for the contents of a disk block.

The storage device is divided into fixed-length blocks, normally 512 words, or 1024 characters long. Blocks are suitable for holding programs or data. They are also used for the Forth system itself.

In Forth, a block storage device is simple and device-independent. Blocks are simply numbered sequentially from 0 to some high number. In order to retrieve a new block, the user types BLOCK, which takes the number he has put on the top of the stack as a block number, reads the block into a buffer, and returns the address of that buffer on top of the stack.

The user can type UPDATE after BLOCK to change the data in a block. Typing FLUSH rewrites updated blocks explicitly.

The Interpreter

The user controls a Forth compute from the terminal. The system is idle and listening for anything from the keyboard until the user types in a complete line. When Forth gets a full line, it attemps to execute the words the user has just typed.

Once the user has debugged a new word, he does not want to type it in again. The Forth text editor lets him store the program in a block. To define a word, or a collection of words, type

n LOAD (n is the block number.)

LOAD is a word that temporarily redirects Forth's text interpreter away from the terminal to the block number the user specifies.

Each block to be loaded must end with the special word ";S", which restores the text interpreter to the source previously in effect.

Before the execution of the application begins, the user must LOAD the disk block that contains the application source test. Thus, the interpreter interprets the source text in disk block. If this block contains definitions, they will be compiled into memory. Any commands defined in this block are available for the user use after the loading takes place.

CHAPTER III

AN IMPLEMENTATION OF FORTH

Hardware

The implementation of the Forth language on Heath H-89 microcompuer system uses the microcomputer and floppy disk to run the program. The system consists of the Heath H-89 microcomputer, the H-77 floppy disk system, and the H-14 printer. The Heath H-89 microcomputer system has 48k bytes of random access memory (RAM). The storage media for the H-77 floppy disk system is the hard-sectored 40-track dislette. Each track has 10 sectors, each sector has 256 formatted bytes. On the floppy disk, there are screens of the Forth source programs. The application programs include the Forth's compiler, termianl handler, and operation system.

The H-14 printer is a computer peripheral that provides computer readouts in a hard copy form. The printer has both upper and lower case characters, full ASCII 96-charcter, and paper feed with adjustable spacing.

Features

The Heath H-89 Forth language version has been implemented by Mr. John Cassadyand, and modified by Kim Harris, and published by the Forth Interest Group. This implementation contains most features of other versions. New paragraph below are listed the features which are implemented on the H-89 microcomputer:

!	Store 16 bits of n at address.
*	Leave the signed product of two signed numbers.
*/	Leave the ratio n4 = n1*n2/n2 where n4 of the operation n1*n2/n3
*/MOD	Leave the quotient n5 and reminder n4 of the operation n1*n2/n3
+	Leave the sum of n1+n2
+!	Add n to the value at the address
+-	Apply the sign of n2 to n1, which is left as n3
-	Leave the difference of n1-n2
/	Leave the signed quotient of n1/n2
/MOD	Leave the remainder and signed quotient of n1/n2
0<	Leave a true flag if the number is less than zero, otherwise leave a false flag.
0=	Leave a true flag if the number is equal to

zero, otherwise leave a false flag.

1+ Increment n1 by 1

2+ Leave n1 incremented by 2

: Used in the form called a colon-definition.

; Terminate a colon-definition and stop further compilation. Compiles the run-time ;S

= Leave a true flag is n1=n2; otherwise leave a false flag.

> Leave a true flag if n1 is greater than n2; otherwise a false flag.

? Print the value contained at the address in free format according to the current base

@ Leave the 16 bit contents of address

ABS Leave the absolute value of n as u.

ALLOT Add the signed number to the dictionary pointer DP

AND Leave the bitwise logical and of n1 and n2 as n3.

BASE A user variable containing the current number base used for input and output conversion

C! Store 8 bits at address. On word addressing computer, further specification is necessary regarding byte addressing.

C@ Leave the 8 bit contents of memory adress. Transmit a carriage return and line feed to the selected output device.

D+ Leave the double number sum of two double numbers

D. Print a signed double number from a 32-bit two's complement value.

DIGIT Converts the ASCII character c to its binary equivalent n2, companied by a true flag.

DMINUS Convert the number to its double number two's complement.

DROP Drop the number from the stack

DUP Duplicate the value on the stack.

FORTH The name of the primary vocabulary. Execution makes Forth the context vocabulary.

HEX Set the numeric conversion base to sixteen (hexademical)

I Used within a DO-LOOP to copy the loop index to the stack.

KEY Leave the ASCII value of the next terminal key stack.

LIST Display the ASCII text of screen n on the selected output device.

MIN Leave the smaller of two numbers.

MINUS Leave the two's complement of a number

MOD Leave the remainder of n1/n2, withthe same sign as n1.

CONSTANT A defining word used in the form: n CONSTANT cccc

DECIMAL Set the numberic conversion BASE for decimal

REPEAT Used within a colon-definition in the form:

BEGIN ... WHILE ... REPEAT

At run-time, REPEAT forces an unconditional branch back to just fter the corresponding BEGIN

UNTIL Occurs within a colon-definition in the form:

BEGIN ... UNTIL

At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN.

DO ... LOOP Set up a finite loop, given the index range

DO ... +LOOP Like DO ... LOOP except adds the value of n (instead of always "1") to the index

IF Occurs is a colon-definition in form:

IF ... ENDIF IF ... ELSE ... END

<BUILD ... DOES> Each time a colon-definition is executed, <BUILD defineds a new word with a high-level execution procedure.

[Suspend compilation

] Resume compilation, to the completion of a colon-defintion

Below are listed the features which are not available on the H-89 microcomputer.

#S Generate ASCII text in the text output buffer

EXECUTE Transfer characters from the terminal to address, until a "return" or the count of charcters have been received.

FENCE A user variable containing an address below which FORGETting is trapped.

LOAD Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S

MESSAGE Print on the selected output device the

text of line n relative to screen 4 of drive 0.

LATEST Leave the name field address of the topmost word in the CURRENT vocabulary.

TIB A user ariable containing the address of the terminal input buffer.

VOCABULARY A defining word used in the form:
VOCABULARY cccc to create a vocabulary definition cccc.

CHAPTER IV

CONCLUSION

Advantage/Disadvantage

Forth is a structured language (as is Pascal) in that it has no GOTO statement labels in the language. Forth object code is extremely compact, even more so than machine language. The reason is that no matter how much work operation performs, each invocation of it takes the same space in the object program. The complete Forth system takes about 7K bytes, and this whole system including the compiler is commonly left in memory as a run-time package. Therefore, 16K bytes and a floppy disk for storing source programs are sufficient hardware for an excellent Forth system.

It is a self-contained operating system. The 7K bytes include terminal and disk handlers and a rudimentary file system. No other software is needed anywhere in the computer.

This language is reentrant so that different users can share the same copy of a program in memory while running at the same time. It is also recursive which means that routines can invoke themselves. The user can type in or load from disk a machine-language routine using

a Forth assembler. Furthermore, the new routine can be executed immediately.

The assembler allows structured conditionals and loops at the machine-code level and it can also assemble unstructured code if desired. Users can define their own macro-intructions, such as being able to use custom-made data types. Since Forth systems have been designed for compatibility, large applications can be moved among very different machine with little change.

Another good feature of Forth is that it puts you in charge of the computer. The user can understand every stop in his software or any desired parts of it. This allows changes to be made easily. The whole system is written in Forth, right down to the bits - the application programs, the compiler, the operating system, and the I/O devices.

Although Forth has many advantages, there are also some disadvantages. The Forth system has no floating-point arithmetic and is not a typed language. For example, if an integer operation is performed on a floating-point quantity, there is no error message printed at compile time or at run time. Another disadvantage of the Forth system is that there is no directory; when a new

disk is inserted, the user or the programmer must know the block numbers for load screeens and data files. Source programs are also difficult to read, but this may be due to lack of familiarity.

Although Forth has some disadvantages, it is still a good programming language. The user needs not worry the parameter passing between the main program and the subroutine. The user can store the words in the dictionary. This language is especially good for team work and big projects. It can speed up the rate of programming. Since this is a relatively new programming language, the program's appearance is strange to the new learner. The use of a Last-In-First-Out stack for parameter passing, and a unique approach to program solving requires the user to reorient his thinking a bit.

Deficiencies Of This Implementation

There are currently several version of the Forth languages. The version used by the Department of Mathematical Sciences is called Fig-Forth. There are some deficiencies in this implementation. Usually the editor is very convenient and efficient to the users. However, the editor was not operational in this system. On other systems, such as the Forth-79 and PDP-11, the user can use the editor to save and correct the text. Without the use of editor, the user cannot save or correct programs or data on disk. The user must retype the whole program or text .

Another deficiency encountered was that the Forth loader did not work. The user could not load screens for execution; therefore, jobs already stored in memory were not available to the user.

APPENDIX

Forth Source Programs Display

```
3 LIST
SCR # 3
  0 ( DEMO TO SHOW FORTH DECISION CAPABILITY )
  1 : CHECK DUP 0=  IF  ." ZERO "  DROP  ELSE
  2     0< IF  ." NEGATIVE"  ELSE  ." POSITIVE"
  3     ENDIF  ENDIF  CR  ;
  4  ;S
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 OK


4 LIST
SCR # 4
  0 ( ERROR MESSAGES )
  1 EMPTY STACK
  2 DICTIONARY FULL
  3 HAS INCORRECT ADDRESS MODE
  4 ISN'T UNIQUE
  5
  6 DISC RANGE ?
  7 FULL STACK
  8 DISC ERROR !
  9
 10
 11
 12
 13
 14 (LP.OPN) IS FALSE!
 15 MOS    SOFTWARE IN FORTH   VERSION 1.1      30-JUL-80
 OK
```

```
5 LIST
SCR # 5
  0 ( ERROR MESSAGES )
  1 COMPILATION ONLY, USE IN DEFINITION
  2 EXECUTION ONLY
  3 CONDITIONS NOT PAIRED
  4 DEFINITION NOT FINISHED
  5 IN PROTECTED DICTIONARY
  6 USE ONLY WHEN LOADING
  7 OFF CURRENT EDITING SCREEN
  8 DECLARE VOCABULARY
  9
 10
 11
 12
 13
 14
 15
 OK
```

```
 6 LIST
 SCR # 6
   0 ( ALTERNATE TERMINAL [ AT: ] HANDLER )
   1 CODE CKAT ( CHECK AT: FOR INPUT CHAR ) ( 335 IN, 1 ANI,
   2  H 0 MVI, L A MOV, HPUSH JMP, ) HEX DB C, OCTAL 335 C, HEX
   3  E6 C, 1 C, 26 C, 0 C, 6F C, C3 C, 328F ,
   4 CODE CLRIF= ( LEAVE 0 ON STACK IF S1LB=S2LB ELSE TOS=S2LB )
   5 ( DE POP, A E MOV, HL POP, H 0 MVI, L CMP, IFZ L H MOV, ENDIF
   6 HPUSH JMP,) D1 C, 7B C, E1 C, 26 C, 0 C, B0 C, 28 C, 01 C, 6C C,
   7  C3 C, 328F ,
   8 (    )
   9 : INAT  (  INPUT CHAR FROM AT: [ ASSUMES ONE READY! ] )
  10     0  NIO@  177  AND  15  CLRIF=  177  CLRIF=  ;
  11 ( ABOVE REMOVES RUBOUTS AND CTL-N [OR] FROM STREAM )
  12 CODE OUTAT ( SEND CHAR TO AT: ) ( DE POP, BEGIN 335 IN, 100
  13 UNTIL A E MOV, 330 OUT, NEXT JMP, ) D1 C, DB C, OCTAL 335 C,
  14 HEX E6 C, 40 C, 28 C, -6 C, 7B C, D3 C, OCTAL 330 C, HEX
  15 HEX C3 C, 3200 ,  OCTAL -->
 OK
```

```
 7 LIST
 SCR # 7
   0 ( TEMPORARY HOME OF AIM-BASED PERSONALIZATION CODE )
   1 HEX : AIM-SEND 4C ( L) OUTAT SEND 20 OUTAT 20 OUTAT CON ;
   2 : AIM-NO WRITE 4E OUTAT COPY ;
   3 : AIM-MORE WRITE 20 OUTAT COPY ;
   4 12003AUD ."  12003AUD" CR
   5 0 NPC !  ;S
   6
   7
   8
   9
  10
  11
  12
  13
  14
  15
 OK
```

```
9 LIST
SCR # 9
  0 ( SPECIAL DEFINITIONS FOR HEATH-BASED SYSTEMS *)
  1 FORTH DEFINITIONS
  2 : OCTAL 8 BASE ! ; BASE @    OCTAL
  3 ( TEMPORARY O->S )   : CODE  CREATE  SMUDGE  ;
  4 CODE O->S ( DE POP, L D MOV, D 0 MVI, H D MOV, DPUSH JMP, )
  5 HEX D1 C, 6A C,  16 C,  0 C,  62 C,  C3 C,
  6 ( DPUSH ) 32BE ,      OCTAL
  7 : S->D  ( SPLIT TO OCTAL )   SWAP 400 * + ;
  8 ( CONVERT WITH PRINTING *)
  9 : <S DUP 0->S . . ; : S> S->D DUP . ;
 10 : FORCE.LP  ( FORCE LP: PRINTOUT BY NULL FILLING *)
 11    P.BUFE LP.BUF @ DO 0 .LP LOOP ;
 12 : CLOSELP ( NORMAL CLOSE OF LP: *)  FORCE.LP CLOSES ;
 13 : LP.TRIAD  ( DOCUMENTATION TO LP: *) LOG TRIAD UNLOG ;
 14 BASE ! ( RESTORE USER BASE )  -->
 15
OK
```

```
10 LIST
SCR # 10
  0 ( TEXT, LINE )
  1 FORTH DEFINITIONS  HEX
  2 : TEXT            ( ACCEPT FOLLOWING TEXT TO PAD *)
  3     HERE C/L 1+  BLANKS  WORD  HERE  PAD  C/L  1+  CMOVE ;
  4
  5 : LINE       ( RELATIVE TO SCR, LEAVE ADDRESS OF LINE *)
  6      DUP  FFF8  AND  17  ?ERROR     ( KEEP ON THIS SCREEN )
  7      SCR  @  (LINE)  DROP  ;
  8 -->
  9
 10
 11
 12
 13
 14
 15
OK
```

```
11 LIST
SCR # 11
  0 (  LINE EDITOR  )
  1 VOCABULARY  EDITOR  IMMEDIATE  HEX
  2 : WHERE             ( PRINT SCREEN # AND IMAGE OF ERROR *)
  3    DUP  B/SCR  /  DUP  SCR  !  ." SCR # "  DECIMAL  .
  4    SWAP  C/L  /MOD  C/L  *  ROT  BLOCK  +  OR  C/L  TYPE
  5    CR  HERE  C@  -  SPACES  5E  EMIT  [COMPILE] EDITOR  QUIT ;
  6
  7 EDITOR  DEFINITIONS
  8 : #LOCATE                  ( LEAVE CURSOR OFFSET-2, LINE-1 *)
  9        R#  @  C/L  /MOD  ;
 10 : #LEAD             ( LINE ADDRESS-2, OFFSET-1 TO CURSOR *)
 11        #LOCATE  LINE  SWAP  ;
 12 : #LAG           ( CURSOR ADDRESS-2, COUNT-1 AFTER CURSOR *)
 13        #LEAD  DUP  >R  +  C/L  R>  -  ;
 14 : -MOVE       ( MOVE IN BLOCK BUFFER ADDR FROM-2, LINE TO-1 *)
 15        LINE  C/L  CMOVE  UPDATE  ;  -->
OK
```

```
12 LIST
SCR # 12
  0 (  LINE EDITING COMMANDS )
  1 : H     ( HOLD NUMBERED LINE AT PAD *)
  2       LINE  PAD  1+  C/L  DUP  PAD  C!  CMOVE  ;
  3
  4 : E     ( ERASE LINE-1 WITH BLANKS *)
  5       LINE  C/L  BLANKS  UPDATE  ;
  6
  7 : S     ( SPREAD MAKING LINE # BLANK *)
  8       DUP  1  -  ( LIMIT )  DO  ( FIRST TO MOVE )
  9       DO  I  LINE  I  1+  -MOVE  -1  +LOOP  E  ;
 10
 11 : D     ( DELETE LINE-1, BUT HOLD IN PAD *)
 12       DUP  H  OF  DUP  ROT
 13       DO  I  1+  LINE  I  -MOVE  LOOP  E  ;
 14
 15 -->
OK
```

```
13 LIST
SCR # 13
  0 (  LINE EDITING COMMANDS )
  1
  2 : N     ( MOVE CURSOR BY SIGNED AMOUNT-1, PRINT ITS LINE *)
  3       R#  +!  CR  SPACE  #LEAD  TYPE  SF  EMIT
  4                          #LAG   TYPE  #LOCATE  .  DROP  ;
  5
  6 : T     ( TYPE LINE GIVE #-1, SAVE ALSO IN PAD *)
  7       DUP  C/L  *  R#  !  DUP  H  0  N  ;
  8
  9 : L     ( RE-LIST SCREEN *)
 10         SCR  @  LIST  0  N  ;
 11 -->
 12
 13
 14
 15
OK
```

```
C'14 LIST
SCR # 14
  0 (  LINE EDITING COMMANDS )
  1 : R      ( REPLACE ON LINE #-1, FROM PAD *)
  2      PAD  1+ SWAP  -MOVE  ;
  3
  4 : P      ( PUT FOLLOWING TEXT ON LINE-1 *)
  5      1  TEXT  R ;
  6
  7 : I      ( INSERT TEXT FROM PAD ONTO LINE # *)
  8      DUP  S  R  ;
  9                                  OR
 10 : TOP     ( MOVE CURSOR TO TOP LEFT OF SCREEN; *)
 11       0  R#  !  ;
 12 —>
 13
 14
 15
OK
```

```
15 LIST
SCR # 15
  0 (  SCREEN EDITING COMMANDS )
  1 : CLEAR              ( CLEAR SCREEN BY NUMBER-1 *)
  2      SCR  !  10  0  DO  FORTH  I  EDITOR  E  LOOP  ;
  3
  4 : COPY        ( DUPLICATE SCREEN-2, ONTO SCREEN-1 *)
  5    B/SCR  *  OFFSET  @  +  SWAP  B/SCR  *  B/SCR  OVER  +  SWAP
  6    DO  DUP  FORTH  I  BLOCK  2  -  !  1+  UPDATE  LOOP
  7    DROP  FLUSH  ;
  8
  9 ( TO BE CHANGED WHEN STRING EDITING ADDED )      ;S
 10
 11
 12
 13
 14
 15
OK
```

```
17 LIST
SCR # 17
  0 ( ***MCS V1.1  30 JUL 83  MODEM PRIMITIVES  )
  1 FORTH DEFINITIONS OCTAL
  2 330  CONSTANT  MIO     5  CONSTANT  MLS
  3
  4 : MIO!  ( STORE S2 @ PORT MIO+S1 )
  5       MIO  +  P!  ;
  6 : MIO@  ( INPUT FROM PORT MIO+S1 )
  7       MIO  +  P@  ;
  8 : SBR   ( SET BAUD RATE -- LSB,MSB  )
  9       200  3  MIO!  1  MIO!  0  MIO!  3  DUP  MIO!  ;
 10
 11 ( INITIALIZE MODEM CHIP )
 12 202  1  SBR  3  4  MIO!
 13 -->
 14
 15
OK
```

```
18 LIST
SCR # 18
  0 ( ALTERNATE TERMINAL [ AT: ] HANDLER )
  1 : CKAT  ( CHECK AT: FOR INPUT CHAR )
  2       MLS  MIO@  1  AND  ;
  3 : CLRIF=  ( IF S1=S2 THEN S1=0 ELSE S1=OLD S2 )
  4       OVER  =  IF  DROP  0  ENDIF  ;
  5 : INAT  (  INPUT CHAR FROM AT: [ ASSUMES ONE READY! ])
  6       0  MIO@  177  AND  15  CLRIF=  177  CLRIF=  ;
  7 ( ABOVE REMOVES RUBOUTS AND CTL-M [CR] FROM STREAM )
  8 : OUTAT ( SEND  CHAR TO AT: )
  9       BEGIN  MLS  MIO@  100  AND  UNTIL  0  MIO!  ;
 10 : 300BAUD 200 1 SBR ;     : 1200BAUD 140 0 SBR ;
 11 : WAITAT BEGIN CKAT UNTIL ;
 12 -->
 13
 14
 15
OK
```

```
19 LIST
SCR # 19
  0 (  DISK INTERFACE FOR STREAM  I/O ON CHL1 )
  1 : CLOSE1 1 CLOSE DROP ;
  2 : CHL1 10000 OFFSET ! ;
  3 ( ***ELIMINATE THIS LINE AND THE NEXT ONE AFTER VER 1.1.1 )
  4 105 377 S->D  CONSTANT  SVSFTH
  5 : OPENRD  ( OPEN FILENAME POINTED TO BY TOS FOR READ )
  6  SVSFTH  0  [ 42 1 S->D ] LITERAL  C.MODS  S.CHEK  ;
  7 : OPENWR  ( OPEN FILENAME POINTED TO BY TOS FOR WRITE )
  8 SVSFTH  0  [ 43 1 S->D ] LITERAL C.MODS  S.CHEK  ;
  9 : CKOUT  .HERR  IF  WARN  ENDIF  ;
 10 -->
 11
 12
 13
 14
 15
OK
```

```
20 LIST
SCR # 20
  0 (  SSD FIFILE PRIMITIVES )
  1 0 VARIABLE CHL1EOF 1 CONSTANT EC.EOF ( REMOVE LATTER LATER )
  2 0    VARIABLE TIME-DELAY : MS TIME-DELAY @ ;
  3 : PAUSE MS 0 DO LOOP ;
  4 ( WAIT FOR PROMPT OF [NPC] CHARS AFTER CR SENT TO AT )
  5 : HALFIX DUP 12 = IF DROP 15 ENDIF ;
  6 76 ( ) ) VARIABLE PROMPT 11 ALLOT  1  VARIABLE NPC
  7 : WAIT-PROMPT PROMPT NPC @ + PROMPT DO BEGIN WAITAT
  8              I   C@  INAT  =  UNTIL  LOOP ;
  9 : SCHAR  C@  -DUP  IF HALFIX  DUP  DUP  OUTAT  EMIT
 10  15  =  IF  NPC  @  IF  WAIT-PROMPT  ENDIF  ENDIF  ENDIF  ;
 11
 12 : SBLOCK  B/BUF  OVER  +  SWAP  DO  I  SCHAR  LOOP ;
 13 : GBLOCK  BLOCK  DISK-ERROR  @  -DUP  IF  EC.EOF  =
 14    IF  1  CHL1EOF  !  ELSE  WARN  ENDIF  ENDIF  ;
 15 -->
OK
```

```
21 LIST
SCR # 21
  0 ( BUFFER HANDLERS )
  1 0  VARIABLE TBUF  10000  ( 4K )  ALLOT  HERE  CONSTANT  BUFEND
  2    TBUF  DUP  2+  SWAP  !
  3
  4 : WBUF1  TBUF  @  SWAP  OVER  C!  1+  DUP  TBUF  !  BUFEND
  5         =  IF  ." BUFFER OVERFLOW!"  CR  WARM  ENDIF  ;
  6 : CLRTE  ( CLEAR TO END OF BUFFER )
  7         TBUF @ BUFEND OVER - ERASE ;
  8 : COMPB  ( COMPUTE NUMBER OF BLOCKS )
  9         TBUF  @  TBUF  2+  -  B/BUF  /MOD  SWAP  IF
 10         1+  ENDIF  B/BUF  *  ;
 11
 12 : SBKOFILE  ( SEND FILE ON SELECTED CHANNEL TO AT: )
 13    0  CHL1EOF  !  -1  BEGIN  1+  DUP  GBLOCK  CHL1EOF
 14   @  0=  IF SBLOCK  ELSE  DROP  ENDIF  CHL1EOF  @
 15    UNTIL  DROP  ;        -->
OK
```

```
22 LIST
SCR # 22
  0 ( CHECK TT: FOR CHAR, GET IT , GET FILENAME )
  1 : SCIN  0  0  0  [ ( .SCIN ) 1  0  S->0  ]  LITERAL
  2         C.HDOS  >R  DROP  DROP  DROP  R>  DUP  1  AND
  3         IF  DROP  0  ELSE  0->S  SWAP  DROP  1  ENDIF  ;
  4 : CKTT  SCIN  >R  R  IF  ( CTL-V )  31  OVER  =
  5         IF  ( MAP CTL-V TO NULL )  DROP  0  ELSE
  6         ( CTL-T )  24  OVER  =  IF  ( MAP CTL-T TO CTL-S )
  7         DROP  ( CTL-S) 23 ENDIF ENDIF ENDIF HALFIX  R> ;
  8
  9 : GFN  ( GET FILENAME  AND SET S1 = START OF STRING )
 10       BL  WORD  HERE  1+  ;
 11
 12 DECIMAL  ." BASE IS DECIMAL"
 13 32  CONSTANT  N1  41  CONSTANT  N2
 14 -->
 15
OK
```

```
23 LIST
SCR # 23
  0 ( USER LEVEL MOS COMMANDS  20-JUL-80)
  1 : CON  ( ENTER CONVERSATION MODE )
  2       BEGIN CKTT IF OUTAT ENDIF CKAT IF INAT -DUP
  3       IF EMIT ENDIF ENDIF 0 UNTIL ;
  4 : SEND  ( SEND FILE OUT )  CR  0  DISK-ERROR !
  5       FLUSH GFN OPENRD CKOUT CHL1 SENDFILE CLOSE1 CHL0 ;
  6 : OPEN  ( OPEN FILE FOR STORAGE PURPOSES )
  7       GFN  OPENWR  CKOUT ;
  8 : COPY  TBUF  DUP  2+  !  BEGIN  WAITAT
  9       INAT DUP EMIT WBUF1 0 UNTIL ;
 10 : HELP  N2  N1  DO  I  MESSAGE  CR  LOOP  ;
 11
 12 : WRITE  CURTE  COMPB 0 SWAP TBUF 2+ SWAP [ ( WRITE )
 13 5 1 S->D ] LITERAL C.MODS S.CHEK .HERR IF WARN ENDIF ;
 14 : CLOSE ( CLOSE OUTPUT FILE ) WRITE CLOSE1 ;
 15 ;S
OK
```

```
24 LIST
SCR # 24
  0 ( BACKUP FOR SCR 22, CHECK TT: FOR CHAR, GET IT, GET FILENAME)
  1 : SCIN  0  0  0  [ ( .SCIN ) 1  0  S->D  ]  LITERAL
  2       C.MODS  >R  DROP  DROP  DROP  R>  DUP  1  AND
  3       IF  DROP  0  ELSE  0->S  SWAP  DROP  1  ENDIF  ;
  4 : CKTT  SCIN  >R  R  IF  ( CTL-Y )  31  OVER  =
  5       IF  ( MAP CTL-Y TO NULL )  DROP  0  ELSE
  6       ( CTL-T )  24  OVER  =  IF  ( MAP CTL-T TO CTL-S )
  7       DROP  ( CTL-S) 23 ENDIF ENDIF ENDIF HALFIX  R> ;
  8
  9 : GFN  ( GET FILENAME  AND SET S1 = START OF STRING )
 10       BL  WORD  HERE  1+ ;
 11
 12 DECIMAL  ." BASE IS DECIMAL"
 13 32  CONSTANT  N1  41  CONSTANT  N2
 14 ;S
 15
OK
```

Forth Sample Programs

This program is using DO-LOOP to yield an ASCII character.

```
8080 FIG-FORTH 1.1
: STRING <BUILDS ALLOT DOES> + ; OK
5 STRING BEANS OK
: STUFF-BEANS 5 0 DO
   I 65 +
   I BEANS C !
   LOOP ; OK
: SPILL-BEANS 5 0 DO
  I BEANS C@
   EMIT ; ;? DEFINITION NOT FINISHED
: SPILL 5 0 DO
  I BEANS C@
   EMIT ; ;? DEFINITION NOT FINISHED
: SPILL 5 0 DO
  I BEANS C@
  EMIT ; LOOP SPACE ; OK
STUFF-BEAN STUFF-BEAN?
:STUFF 5 0 DO  :STUFF?
 : AGE 5 0 DO 1 65 + 1 BEANS C! LOOP ; OK
: OLD 5 0 DO I BEANS C@ EMIT LOOP SPACE ; OK
AGE OK
OOLD ABCDE OK
```

This program shows how to use the BUILDS ... DOES to store strings

```
: 2* S 2 * ; OK
: GOOD <BUILDS SMUDGE ] DOES>
  SWAP 2* + @ EXECUTE ; OK
: TEST ."BEAR^"; ."BEAR";?
: TEST ." BEAT " ; OK
TE: TEST1 ." CAT " ; OK
: TEST2 ."DOG DOG " ; OK
GOOD TOTAL TEST TEST1 TEST2
; OK
TEST BEAT OK
1TEST1 CAT OK
TEST2 DOG OK
LIST 1 LIST
```

This program is using the user-defined array to store numbers.

```
: ARRAY
 <BUILDS 0 DO. . LOOP DODOES>
  SWAP 2 * + @ ; OK
  32 24 15 30 17 18 25 OK
  7 ARRAY RANDOM OK
 : RANDOM1 7 -1 RAMDONDOM ; OK
1 RANDOM1 OK
1 RANDOM1 . RANDOM1?
1 RANDOM1 . 265609 OK
2 RANDOM1 . 26509 OK
1 RANDOM . 18 OK
2 RANDOM . 17 OK
3 RANDOM . 30 OK
4 RANDONM . 15 OK
5 RANDOM . 24 OK
6 RANDOM . 32 OK
7 RANDOM . 21127 OK
0 RANDOM . 25 OK
```

This program uses the VARIABLE to store the number.

```
0 TIME VARIABLE TIME  OK
: GOOD  345 TIME !
    0= IF 5 TIME +!
    ELSE 56 TIME +! ; ;? DEFINITION NOT FINISHED
: GOOD 345 TIME !
   0= IF 5 TIME +!
    ELSE 56 TIME +!
    THEN 344 * . ;;   OK
 GOOD . GOOD? EMPTY STACK
  TIME ? 401 OK
 0 VARIABLE AT  OK
  * 88AT 8888 AT TIME ! OK
 TIME . 20248 OK
```

This program is to count the polymonial :

5x**6 -7x**2 +3x - 1 by using the SWAP and DUP

```
: POLY
  DUP DUP DUP DUP DUP DUP
  * * * * *
 5 * SWAP DUP DUP * 7 *
 SWAROT SWAP - SWAP * 3 + 1 - ; OK
2 POLY . 586 OK
3 POLY . 10748 OK
5 POLY . -3464 OK
9 POLY . -10896 OK
10 POLY . -10966 OK
34 POLY . 26570 OK
-9 POLY . 10900 OK
-10 POLY . 10970 OK
-8 POLY . 3586 OK
-7 POLY . 13456 OK
```

References

1. Ewing, Martin S., The Caltech Forth Manual, Second Edition, Owens Valley Radio Observatory, California Institute of Technology, Pasadena, California, 1978
2. Rather, E., Brodie L., Using Forth Forth-79 Standard Edition, Forth, Inc. Hermosa Beach, Ca.
3. Moore, Charles H. "The Evolution of Forth, An Unusual Language" Byte, August, 1980, p76
4. James, John S. "What's Forth? A Turorial Introduction" Byte, August, 1980, p 100
5. Harris, Kim, "Forth Extensibility" Byte, August, 1980, p 164
6. James, John S. PDP-11 Forth User's Guide, September, 1979